User Manual

Programs of dashSVD Algorithm with Shifted Power Iteration for Sparse Data

Xu Feng, Wenjian Yu, et al.

March, 2024

Contents

1 Introduction

This is the user manual of the programs of dashSVD algorithm with shifted power iteration for sparse data. The programs are implemented both in Matlab and in C. The details of algorithm are included in the relevant paper.

2 Efficient Implementation of dashSVD

In this section, we first briefly introduce the Math Kernel Library (MKL) [1], which is an efficient library for matrix computations. Then, we present the details of dashSVD (Alg. 5) program written in C based on MKL. Finally, we provide the program interface in both C and Matlab.

2.1 Overview of MKL

MKL is developed by Intel for high-performance implementation of linear algebra computation [1]. MKL not only provides the parallel implementations of functions in BLAS [6] and LAPACK [7], but also supports some sparse matrix related computations , like the multiplication of a sparse matrix and a dense matrix. Therefore, based on MKL, we can implement dashSVD to obtain a C program with good parallel efficiency.

2.2 Implementation Details of dashSVD

To improve the performance of dashSVD, we implement it in C language with MKL and multi-thread computing. Several careful treatments are imposed in the implementation for better performance on time and parallel efficiency with less memory usage.

Firstly, we reduce the peak memory usage by allocating the space of matrices when they are computed and freeing the space of matrices immediately when they are not needed. Secondly, the following treatments are applied to implement dashSVD with better parallelization.

- We use the high-performance parallel implementations of matrix operations in MKL as much as possible. Therefore, the matrix-matrix, matrix-vector and vector-vector operations are naturally parallelized. We use the following functions of MKL/LAPACK/BLAS to do the main computations in Alg. 5.

  - `mkl_sparse_d_mm` : Compute the multiplication of a sparse matrix stored in compressed sparse row (CSR) format with a dense matrix, which is mainly used in Step 2, 4 and 9 in Alg. 5.
  - `cblas_dgemm` : Compute the matrix-matrix multiplication of two dense matrices, which is mainly used in "eigSVD" (Alg. 3) invoked in Step 2, 4 and 10 of Alg. 5, and the computation of **V** in Step 11 of Alg. 5.
  - `LAPACKE_dsyevd` : Compute the EVD of a dense matrix, which is used in "eigSVD".

- We parallelize the copy operations of matrix and vectors with OpenMP following the implementation in [5] [2]. Therefore, all copy operations in dashSVD are well parallelized with less runtime.

2.3 Interface of dashSVD

In this subsection, we describe the most important functionality and features of dashSVD using the C interface. Firstly, the sparse matrix should be read in as `mat_csr` structure, which includes the following members.

- `nnz`, `nrows` and `ncols`: the numbers of nonzeros, rows and columns of the sparse matrix.
- `values`: the array of length `nnz` storing the values of nonzero elements.
- `cols`: the array of length `nnz` storing the column indices of nonzero elements. The column index ranges from 1 to `ncols`.
- `pointerI`: the array of length `nrows`+1 storing the indices of each row's starting position in `cols`. For example, the column indices of nonzero elements in the *i*-th row are stored as `cols[pointerI[i-1]]` through `cols[pointerI[i]]`.

Parameters `cols`, `pointerI` are designed following the CSR format in MKL. Besides, the structure `mat` is used to store the dense matrix in column major format, including the following members.

- `nrows` and `ncols`: the numbers of rows and columns of the dense matrix.
- `d`: the array storing the value of matrix in size `nrows` $\times$ `ncols` in column-major format.

After specifying the input sparse matrix stored in CSR format, we can call the following functions to run the dashSVD algorithm:

```
dashSVD(A, U, S, V, k)
dashSVD_opt(A, U, S, V, k, p, s, tol)
```

Here, `A` is the input matrix in structure `mat_csr`. `U`, `S` and `V` are the matrices in structure `mat`. Their dimensions are `A.nrows` $\times$ `k`, `k` $\times$ 1 and `A.ncols` $\times$ `k`, respectively. `k` is the target rank of truncated SVD. Function `dashSVD_opt` provides an option for user to specify some parameters, where the arguments `p`, `s` and `tol` are p_{max}, s and tol in Alg. 5 respectively. For function `dashSVD`, the values of `p`, `s` and `tol` are fixed, i.e. 1000, `k`/2 and 10^{-2}, respectively.

We also provide the interface of dashSVD in Matlab:

```
[U, S, V] = dashSVD(A, k, p, s, tol)
```

Here, the arguments `p`, `s` and `tol` can be blank. In that case, the default values 1000, `k`/2 and 10^{-2} are used.

3 Source Files for the Main algorithms and Drivers

3.1 Main Algorithms

- `Src/matlab/exxamples/basic_rSVD_shift.m`: basic randomized SVD algorithm with shifted power iteration and shift updating scheme (Alg. 2 without accelerating skills).
- `Src/matlab/src/dashSVD.m`: dashSVD with shifted power iteration for sparse data in Matlab (dashSVD with accelerating skills).
- `Src/mkl/src/dashsvd.c`: dashSVD algorithm in our paper which is implemented in C with MKL and OpenMP.

3.2 Drivers in Matlab

- `Src/matlab/example/ShiftedPowerIteration_Test.m`: used to test the effectiveness of shifted power iteration. The comparison is between Alg. 1 (`basic_rSVD.m`) [3], Alg. 2* (`basic_rSVD_shift_noupdate.m`) and Alg. 2 (`basic_rSVD_shift.m`) on Dense1 or Dense2.
- `Src/matlab/example/dashSVD_Test.m`: used to test the effectiveness of dashSVD compared with LanczosBD in `svds` in Matlab on SNAP. (Notice that `svdstest.m` and `LanczosBDtest.m` is the modification of `svds` to make sure LanczosBD can produce results with settled times of restarting.)

- `Src/matlab/example/AccuracyControl_Test.m`: used to validate the PVE criterion of accuracy control used in dashSVD.

3.3 Drivers in C

The program for testing dashSVD with MKL is in "`Src/mkl/example`". The MKL library needs the support of oneAPI in Intel [1]. When the libraries MKL and OpenMP have been installed, firstly modified the path of MKL in the makefile("`Src/mkl/examples/makefile`"), and secondly use "make"

```
>> make
```

to produce the executable program "dashsvdtest". Developers can use

```
>> OMP_NUM_THREADS=8 OMP_PROC_BIND=close OMP_PLACES=cores ./
   dashsvdtest
```

to run the driver with one NUMA domain for comparison in one socket with 8 threads. The result of program is an example of testing the dataset SNAP [4] on dashSVD with and without accuracy control.

4 Additional Notes

- Because the source codes in C rely on not only the general functions in BLAS and LAPACK but also the function `mkl_sparse_d_mm` to compute the multiplication of a sparse matrix stored in CSR format with a dense matrix only in MKL, they are not able to be complied directly with other versions of BLAS and LAPACK.
- When compiled on MacOS, one should download the older version of oneAPI to install MKL, because from oneAPI version 2024 on macOS is not supported. Besides, OpenMP should be prepare when the compile clang dose not support OpenMP. Then, use clang instead of gcc as the compiler to compile the source codes with correct path of links.

References

[1] Intel oneAPI Math Kernel Library. https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html, 2021.

[2] Randqr. https://users.oden.utexas.edu/~pgm/Codes/randqb_codes_intel_mkl.zip, 2021.

[3] Halko, N., Martinsson, P. G., and Tropp, J. A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review 53*, 2 (2011), 217–288.

[4] Leskovec, J., and Krevl, A. SNAP datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[5] Martinsson, P. G., and Voronin, S. A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices. *SIAM J. Sci. Comput 38* (2016), S485––S507.

[6] Netlib. Basic linear algebra subprograms. `http://www.netlib.org/blas/`, 2022.

[7] Netlib. Linear algebra package. `http://www.netlib.org/lapack/index.html`, 2022.